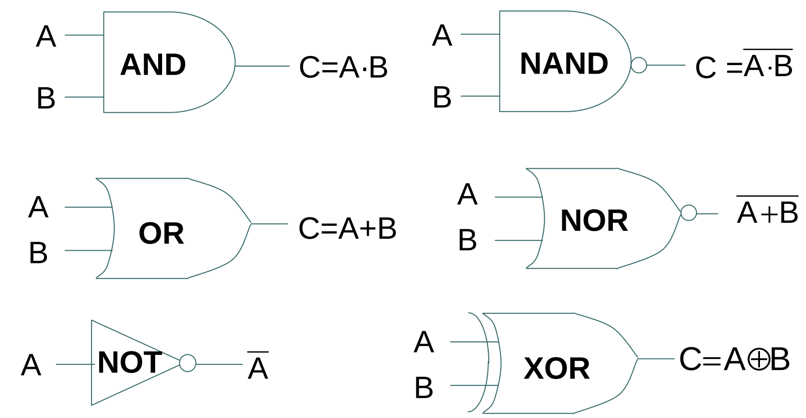
Lecture 14

Today we will

- Learn how to implement mathematical logical functions using logic gate circuitry, using
  - Sum-of-products formulation
  - NAND-NAND formulation
- Learn how to simplify implementation using
  - Boolean algebra
  - Karnaugh maps

Logic Gates



(EXCLUSIVE OR)

Properties of Logic Functions

- These new functions, AND, OR, etc., are mathematical functions just like +, -, sin(), etc.
- The logic functions are only defined for the domain {0, 1} (logic functions can only have 0 or 1 as inputs).
- The logic functions have range {0, 1} (logic functions can only have 0 or 1 as outputs)
- AND acts a lot like multiplication.
- OR acts a lot like addition.
- Learn the properties so you can simplify equations!

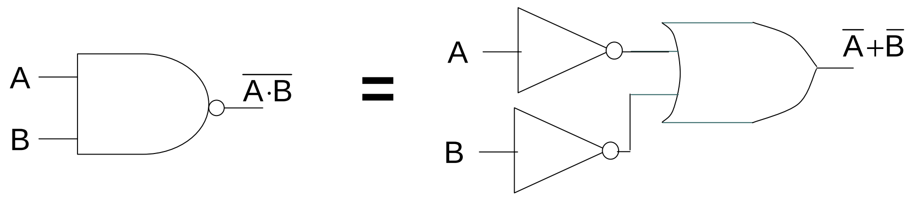
Properties of Logic Functions

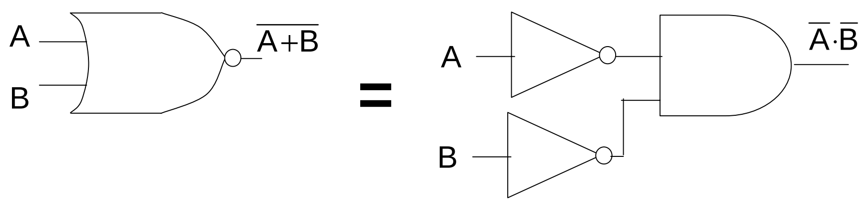
$A + 0 = A$	$A \bullet 1 = A$
$A + \overline{A} = 1$	$A \bullet \overline{A} = 0$
$A + A = A$	$A \bullet A = A$
$A + B = B + A$	$A \bullet B = B \bullet A$
$A + (B + C) = (A + B) + C$	$(A \bullet B) \bullet C = A \bullet (B \bullet C)$
$A \bullet (B + C) = A \bullet B + A \bullet C$	$A + B \bullet C = (A + B) \bullet (A + C)$
$A + A \bullet B = A$	$A \bullet (A + B) = A$
DeMorgan's Law:	$\overline{A \bullet B} = \overline{A} + \overline{B}$
	$\overline{A} \bullet \overline{B} = \overline{A + B}$

De Morgan's Law

$\overline{A \cdot B} = \overline{A} + \overline{B}$

$\overline{A} \cdot \overline{B} = \overline{A + B}$





Logical Synthesis

- Suppose we are given a truth table or Boolean expression defining a mathematical logic function.
- Is there a method to implement the logical function using basic logic gates?
- One way that always works is the "sum of products" formulation. It may not always be the best implementation for a particular purpose, but it works.

Sum-of-Products Method

1. Create a **Boolean expression** for the function in **sum-of-products** form.

   This means represent the function **F** by groups of ANDed inputs (products) that are then ORed together (sum of products).

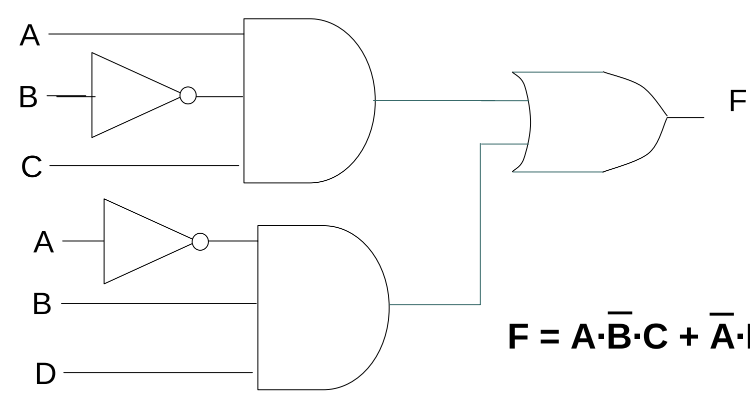
F = A·B·C + A·B·D — **is** in sum-of-products form

F = A·B·(C + D) — **is not** in sum-of-products form

- **How to get to sum-of-products form?**
- Use properties to manipulate given Boolean equation
- Look at each "1" in truth table, write product of inputs that creates this "1", OR them all together

Sum-of-Products Method

2. Implement sum-of-products expression with one stage of inverters, one stage of ANDs, and one big OR:



$\mathbf{F = A \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot D}$

Example (Adder)

A	B	C	S_1	S_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Input | Output

S_1 using sum-of-products:

1) Find where S_1 is "1"

2) Write down product of inputs which create each "1"

$\overline{A}BC$ $A\overline{B}C$

$AB\overline{C}$ ABC

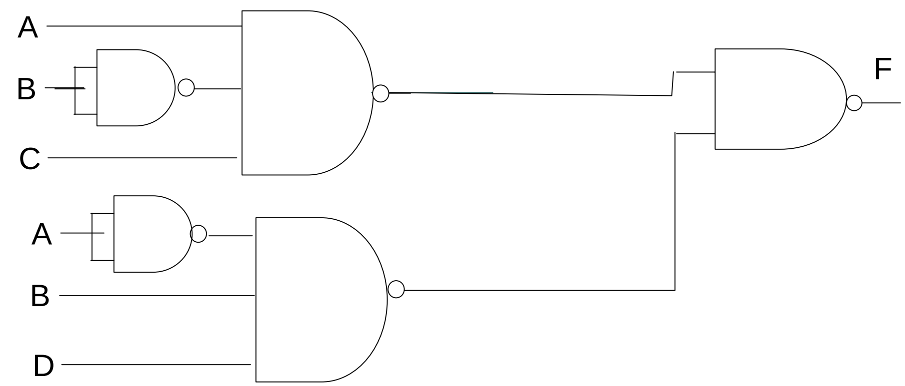
3) Sum all products

$\overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

4) Draw circuit

NAND-NAND Implementation

- We can easily turn our sum-of-products circuit into one that is made up solely of NANDs (generally cheaper):



Karnaugh Maps

To find a simpler sum-of-products expression,

Write the truth table of your circuit into a special table.

A \ B	0	1
0		
1		

2 Inputs

A \ BC	00	01	11	10
0				
1				

3 Inputs

AB \ CD	00	01	11	10
00				
01				
11				
10				

4 Inputs

For each "1", circle the biggest 2m by 2n block of "1's" that includes that particular "1".

Write the product that corresponds to that block, and finally sum.

Example (Adder)

A	B	C	S_1	S_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Input | Output

Simplification for S_1:

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1